案例 10-区域四邻接点边界填充算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

说明：本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

一、知识点

1. 四邻接点

对于多边形区域内部的任意一个种子像素，其左、上、右、下这四个像素称为四邻接点。

2. 种子填充算法

从种子像素点开始，使用四邻接点方式搜索下一像素点的填充算法称为四邻接点种子填充算法；种子填充算法一般要求区域边界与内部填充色不同，输入参数只有种子坐标位置和填充颜色，一般需要使用堆栈数据结构来实现。

3. 扫描线种子填充算法

扫描线种子填充算法的原理是先将种子像素入栈，种子像素为栈底像素，然后判断栈是否为空，如果不为空，执行下列操作：

(1)栈顶像素出栈

(2)沿扫描线对出栈像素的左右像素进行填充，直到遇到边界像素为止。即每出栈一个像素，就对区域包含该像素的整个连续区间进行填充。

(3)同时记录该区域，将区域最左端像素记为 X_{left}，最右端像素记为 X_{right}。

(4)在区间[X_{left}，X_{right}]中检查与当前扫描线相邻的上下两条扫描线的有关像素是否全为边界像素或已填充像素，若有非边界且未填充的像素，则把未填充区间的最右端像素取作种子像素入栈。

二、案例描述

本案例用区域四邻接点边界填充算法，填充“好日子”字样。

三、实现步骤

1. 添加栈结点 CStackNode 类。
2. 在 CTestView 类中定义出/入栈函数、边界填充函数以及双缓冲绘图函数。
3. 在 CTestView 类的 BoundaryFill 函数中使得上下左右四个像素入栈。
4. 在 CTestView 的构造函数中对种子颜色，边界颜色和背景颜色进行初始化。
5. 在 DoubleBuffer 函数中绘图并在 OnDraw 函数中进行调用。

四、主要算法

CTestView 类

```
public:
  void DoubleBuffer(CDC* pDC);//双缓冲绘图
  void BoundaryFill();        //边界填充算法
  void Push(CPoint point);    //入栈
  void Pop(CPoint &point);    //出栈
protected:
  int nClientWidth, nClientHeight;//屏幕客户区宽度和高度
  int nHWidth, nHHeight;//屏幕客户区的半宽和半高
  COLORREF SeedClr, BoundaryClr, BkClr;//种子颜色、边界颜色和背景色
  CStackNode* pHead, *pTop;//结点指针
  CPoint Seed, Left, Top, Right, Bottom;//种子及其四个邻接点
void CTestView::DoubleBuffer(CDC* pDC)
{
  CRect rect;//定义客户区
  GetClientRect(&rect);//获得客户区的大小
  nClientWidth = rect.Width();//屏幕客户区宽度
  nClientHeight = rect.Height();//屏幕客户区高度
  nHWidth = nClientWidth / 2;//屏幕客户区半宽
  nHHeight = nClientHeight / 2;//屏幕客户区半高
  CDC memDC;
  memDC.CreateCompatibleDC(pDC);
  CBitmap NewBitmap, *pOldBitmap;
  NewBitmap.LoadBitmap(IDB_BITMAP1);
  pOldBitmap = memDC.SelectObject(&NewBitmap);
  BITMAP bmp;
  NewBitmap.GetBitmap(&bmp);
  int nX = rect.left + (nClientWidth - bmp.bmWidth) / 2;
              //计算位图在客户区的中心点
  int nY = rect.top + (nClientHeight - bmp.bmHeight) / 2;
```

```
  pDC->BitBlt(nX, nY, nClientWidth, nClientHeight, &memDC, 0, 0, SRCCOPY);
  memDC.SelectObject(pOldBitmap);
  NewBitmap.DeleteObject();
}
void CTestView::BoundaryFill()//四邻接点边界填充算法
{
  CDC* pDC = GetDC();
  pHead = new CStackNode; //建立栈头结点
  pHead->pNext = NULL;//栈头结点的指针域为空
  Push(Seed);              //种子像素入栈
  if (BkClr != pDC->GetPixel(Seed.x, Seed.y) && BoundaryClr
                != pDC->GetPixel(Seed.x, Seed.y))
  {
      while (NULL != pHead->pNext)//如果栈不为空
      {
          CPoint PopPoint;
          Pop(PopPoint);
          if (SeedClr == pDC->GetPixel(PopPoint.x, PopPoint.y))
              continue;//加速处理
          pDC->SetPixelV(PopPoint.x, PopPoint.y, SeedClr);
          Left.x = PopPoint.x - 1;//搜索出栈结点的左方像素
          Left.y = PopPoint.y;
          COLORREF CurPixClr;     //当前像素的颜色
          CurPixClr = pDC->GetPixel(Left.x, Left.y);
          if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
                      //不是边界色并且未置成填充色
              Push(Left);//左方像素入栈
          Top.x = PopPoint.x;//搜索出栈结点的上方像素
          Top.y = PopPoint.y + 1;
          CurPixClr = pDC->GetPixel(Top.x, Top.y);
          if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
              Push(Top);//上方像素入栈
          Right.x = PopPoint.x + 1;//搜索出栈结点的右方像素
          Right.y = PopPoint.y;
          CurPixClr = pDC->GetPixel(Right.x, Right.y);
          if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
              Push(Right);//右方像素入栈
          Bottom.x = PopPoint.x;//搜索出栈结点的下方像素
          Bottom.y = PopPoint.y - 1;
          CurPixClr = pDC->GetPixel(Bottom.x, Bottom.y);
          if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
              Push(Bottom);//下方像素如栈
      }
  }
```

```
    else
        MessageBox(_T("请在空心字体内部单击鼠标左键!"), _T("提示"));
    delete pHead;
    pHead = NULL;
    ReleaseDC(pDC);
}
void CTestView::Push(CPoint point)//入栈函数
{
    pTop = new CStackNode;
    pTop->PixelPoint = point;
    pTop->pNext = pHead->pNext;
    pHead->pNext = pTop;
}
void CTestView::Pop(CPoint &point)//出栈函数
{
    if (pHead->pNext != NULL)
    {
        pTop = pHead->pNext;
        pHead->pNext = pTop->pNext;
        point = pTop->PixelPoint;
        delete pTop;
    }
}
```

五、实现效果

区域四邻接点边界填充算法效果如图 10-1 所示。

图 10-1 区域四邻接点边界填充算法效果图

六、遇到的问题及解决方案

四邻结点种子填充和八邻接点种子填充的区别：

四邻接点种子填充算法和八邻接点种子填充算法的本质区别还是四邻结点和八邻接点的区别；四邻结点有四个方向，八邻接点有八个方向，可以通过左上、右上、左下、右下这四个方向通过并填充狭窄区域。

七、案例心得

四连通区域算法填充是利用四邻接点填充算法填充的，对图片中的边界和背景颜色初始化，确定填充区域，再对种子颜色进行初始化，以完成对字样空心区域的填充；由于“好日子”字样没有狭窄区域，所以四邻结点种子填充算法可以进行填充，但是如果字样出现了狭窄区域则四邻结点种子填充算法就无法进行填充了，需要使用八邻结点种子填充算法。该填充使用的是扫描线种子填充算法，利用扫描线种子填充算法对每一个区间只保留其最右端像素做为种子像素入栈，很大程度上减小了栈空间，提高了填充效率。